Eloquent Basics

No SQL statment but PHP code

Laravel's Eloquent ORM

What is Eloquent?

- Laravel's built-in ORM
- **Most popular PHP ORM**
- **Elegant syntax** (hence the name "Eloquent")
- **Powerful features** out of the box

Eloquent Features:

- Automatic SQL generation
- Built-in validation
- Relationship handling
- Query builder
- Mass assignment protection
- Soft deletes
- Database events

CRUD Operations Comparison

CREATE

PHP Create:

```php
function create_student() {
    $input = getRequestData();
    $students = load_students();
    $new_id = get_next_id($students);
    $new_student = new Student();
    $new_student->setId($new_id);
    $new_student->setName($input['name'] ?? '');
    $new_student->setEmail($input['email'] ?? '');
    $new_student->setMajor($input['major'] ?? '');
    $new_student->setYear($input['year'] ?? 1);
    $students[] = $new_student->toArray();
    save_students($students);
    // Return response...
}
```

Eloquent Create (Method 1):

```php
$student = new Student();
$student->name = 'John Doe';
$student->email = 'john@example.com';
$student->major = 'Computer Science';
$student->year = 2;
$student->save();
```

Eloquent Create (Method 2):

```php
$student = Student::create([
    'name' => 'John Doe',
    'email' => 'john@example.com',
    'major' => 'Computer Science',
    'year' => 2
]);
```

4 lines vs 40+ lines!

Read Operations

PHP Read:

```php
// Database connection
$pdo = new PDO("mysql:host=localhost;dbname=database", $username, $password);
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// Get all students
// Laravel: $students = Student::all();
$stmt = $pdo->prepare("SELECT * FROM students");
$stmt->execute();
$students = $stmt->fetchAll(PDO::FETCH_ASSOC);

// Find by primary key
// Laravel: $student = Student::find(1);
$stmt = $pdo->prepare("SELECT * FROM students WHERE id = ?");
$stmt->execute([1]);
$student = $stmt->fetch(PDO::FETCH_ASSOC);
```

Eloquent Read:

```php
// Get all students
$students = Student::all();

// Find by primary key
$student = Student::find(1);
```

Update Operations

PHP Update:

```php
function update_student($id) {
    $input = getRequestData();
    $students = load_students();

    for ($i = 0; $i < count($students); $i++) {
        if ($students[$i]['id'] == $id) {
            if (isset($input['name'])) $students[$i]['name'] = $input['name'];
            if (isset($input['email'])) $students[$i]['email'] = $input['email'];
            $students[$i]['updated_at'] = date('Y-m-d H:i:s');
            save_students($students);
            return;
        }
    }
}
```

25+ lines of complex logic!

Eloquent Update:

```php
// Method 1: Find and update
$student = Student::find(1);
$student->name = 'New Name';
$student->save();

// Method 2: Update directly
$student = Student::find(1);
$student->update([
    'name' => 'New Name',
    'email' => 'new@email.com'
]);

// Method 3: Update multiple records
Student::where('year', 1)->update(['year' => 2]);
```

Laravel automatically handles `updated_at` timestamp!

Delete Operations

Your Manual Delete:

```
function delete_student($id) {
    $students = load_students();

    for ($i = 0; $i < count($students); $i++) {
        if ($students[$i]['id'] == $id) {
            $deleted_student = $students[$i];
            array_splice($students, $i, 1);
            save_students($students);
            // Return response...
            return;
        }
    }
    // Handle not found...
}
```

Eloquent Delete:

```php
// Delete by finding first
$student = Student::find(1);
$student->delete();

// Delete without retrieving
Student::destroy(1);

// Delete multiple
Student::destroy([1, 2, 3]);

// Delete with conditions
Student::where('year', '<', 1)->delete();
```

Working with Collections

- A Laravel Collection is a powerful wrapper for working with arrays of data.
- It provides a fluent, convenient interface for manipulating data with dozens of helpful methods.

```php
$data = collect([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);

// Count items
$count = $data->count();  // 10

// Check if contains value
$containsFive = $data->contains(5);  // true

// Sum all values
$sum = $data->sum();  // 55
```

Eloquent Returns Collections:

```php
$students = Student::all();  // Returns Collection, not array

// Collection methods (Laravel magic!)
$names = $students->pluck('name');           // Get just names
$csStudents = $students->where('major', 'CS'); // Filter
$count = $students->count();                 // Count
$grouped = $students->groupBy('major');      // Group by field

// Convert to array if needed
$array = $students->toArray();
$json = $students->toJson();
```

Pluck

- pluck() is a Collection method that extracts specific values from a collection of arrays or objects.
- Think of it as "picking" or "pulling out" one field from each item in your collection.

PHP Equivalent:

```php
$students = load_students();
$names = [];
foreach ($students as $student) {
    $names[] = $student['name'];
}
```

Laravel `pluck()`

id	name	email
1	Alice	alice@example.com
2	Bob	bob@example.com
3	Carol	carol@example.com

- Without pluck -> Returns a collection of Student objects

```
$students = Student::all();
```

- With pluck('name') -> Returns: ["Alice", "Bob", "Carol"]

```
$names = $students->pluck('name');
```

Eloquent vs PHP Queries

Retrieve All Students:

```php
// PHP way:
function load_students() {
    $file_path = 'data/students.json';
    if (!file_exists($file_path)) return [];
    $json_data = file_get_contents($file_path);
    return json_decode($json_data, true) ?: [];
}

// Eloquent way:
$students = Student::all();
```

Find Specific Student:

```php
// PHP way: Loop through the array, check IDs
function findStudentById($id) {
    // Simulate database connection
    $pdo = new PDO('mysql:host=localhost;dbname=university', $username, $password);

    // Raw SQL query
    $stmt = $pdo->prepare("SELECT * FROM students WHERE id = ?");
    $stmt->execute([$id]);
    $student = $stmt->fetch(PDO::FETCH_ASSOC);

    return $student;
}

// Eloquent way:
$student = Student::find(1);                    // Find by ID
```

```php
// PHP way:
function findStudentByEmail($email) {
    $pdo = new PDO('mysql:host=localhost;dbname=university', $username, $password);

    // Raw SQL with email search
    $stmt = $pdo->prepare("SELECT * FROM students WHERE email = ?");
    $stmt->execute([$email]);
    $student = $stmt->fetch(PDO::FETCH_ASSOC);

    return $student;
}

// Eloquent way:
$student = Student::where('email', 'john@email.com')->first(); // Find by email
```

Query Builder Methods

Powerful Query Building:

```php
// Chain methods for complex queries
$students = Student::where('year', '>', 1)
                   ->where('major', 'like', '%Computer%')
                   ->orderBy('name')
                   ->take(5)
                   ->get();

// Conditional queries
$query = Student::query();

if ($major) {
    $query->where('major', $major);
}

if ($year) {
    $query->where('year', $year);
}
```

Complex Queries Made Simple:

```php
// Find all Computer Science students in year 2 or higher
$students = Student::where('major', 'Computer Science')
                   ->where('year', '>=', 2)
                   ->orderBy('name')
                   ->get();

// Get student count by major
$counts = Student::groupBy('major')
                 ->selectRaw('major, count(*) as total')
                 ->get();

// Find students with specific email domain
$students = Student::where('email', 'like', '%@university.edu')->get();
```

Error Handling

Manual SQL Errors (Frequent and Dangerous):

```
// Hard to debug SQL errors
$sql = "SELCT * FROM students";  // Typo!
$result = $mysqli->query($sql);  // Generic error message

// SQL injection vulnerability
$id = $_GET['id'];
$sql = "SELECT * FROM students WHERE id = $id";  // Dangerous!
```

ORM Error Handling (Clear, Safe, and Helpful):

```php
// Clear, helpful error messages
try {
    $student = Student::findOrFail($id);
} catch (ModelNotFoundException $e) {
    return response()->json(['error' => 'Student not found'], 404);
}

// Automatic SQL injection protection
$student = Student::where('id', $userInput)->first();  // Always safe!
```